

NAVAL POSTGRADUATE SCHOOL MONTEREY, CALIFORNIA



THESIS

SIMULATION AND ANALYSIS OF PREDICTIVE READ CACHE (PRC) PERFORMANCE

by

Altay Çamlıgüney

September 1996

Advisor:

Douglas J. Fouts

Approved for public release; distribution is unlimited.

19970219 023

19970219 023

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 1996		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE SIMULATION AND ANALYSIS OF PREDICTIVE READ CACHE (PRC) PERFORMANCE			5. FUNDING NUMBERS	
6. AUTHOR(S) Altay Çamlıgüney				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) Memory subsystem bandwidth and latency are two major problems for modern computer architectures because memory speed should grow linearly with CPU speed to maintain balanced system performance. However, in recent years, CPU speed has increased much more rapidly than memory speed. One common approach to memory system design, which can provide remarkable speedup, is to use one or more fast cache memories in a hierarchical architecture. The Predictive Read Cache is an alternative to second-level cache memories with its fast, simple and inexpensive design. Previous studies show that a small predictive read cache can give a better performance improvement than a much larger second-level cache for an Intel 486 microprocessor with a very small cost. This thesis continues previous efforts in design and optimization of the predictive read cache. Using address trace data from a SPARCstation 1+ running Sun-OS and the SPEC benchmarks, the simulations demonstrate that a small predictive read cache can give even better performance improvements than the previous results obtained using 486 trace data. Since SPARC is the most common RISC architecture today, it was of great importance to simulate the PRC by using address traces captured from a SPARC-based platform. The predictive read cache is ideal for embedded systems in space-based, weapons-based and portable computing applications that utilize advanced RISC processors.				
14. SUBJECT TERMS Cache, Predictive, Memory			15. NUMBER OF PAGES 56	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

Approved for public release; distribution is unlimited.

**SIMULATION AND ANALYSIS OF PREDICTIVE READ CACHE (PRC)
PERFORMANCE**

Altay Çamlıgüney
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1990

Submitted in partial fulfillment
of the requirements for the degree of

**MASTER OF SCIENCE
IN
ELECTRICAL AND COMPUTER ENGINEERING**

from the

NAVAL POSTGRADUATE SCHOOL

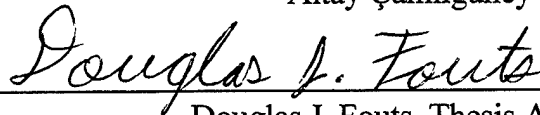
September 1996

Author:



Altay Çamlıgüney

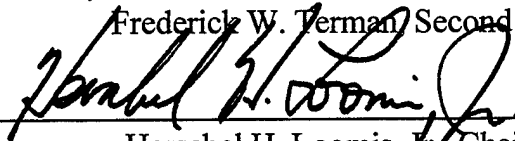
Approved by:



Douglas J. Fouts, Thesis Advisor



Frederick W. Terman, Second Reader



Herschel H. Loomis, Jr., Chairman

Department of Electrical and Computer Engineering

ABSTRACT

Memory subsystem bandwidth and latency are two major problems for modern computer architectures because memory speed should grow linearly with CPU speed to maintain balanced system performance. However, in recent years, CPU speed has increased much more rapidly than memory speed. One common approach to memory system design, which can provide remarkable speedup, is to use one or more fast cache memories in a hierarchical architecture. The Predictive Read Cache is an alternative to second-level cache memories with its fast, simple and inexpensive design. Previous studies show that a small predictive read cache can give a better performance improvement than a much larger second-level cache for an Intel 486 microprocessor with a very small cost. This thesis continues previous efforts in design and optimization of the predictive read cache. Using address trace data from a SPARCstation 1+ running Sun-OS and the SPEC benchmarks, the simulations demonstrate that a small predictive read cache can give even better performance improvements than the previous results obtained using 486 trace data. Since SPARC is the most common RISC architecture today, it was of great importance to simulate the PRC by using address traces captured from a SPARC-based platform. The predictive read cache is ideal for embedded systems in space-based, weapons-based and portable computing applications that utilize advanced RISC processors.

TABLE OF CONTENTS

I. INTRODUCTION	1
A. MEMORY HIERARCHY.....	1
B. CACHE MEMORIES.....	2
C. PREDICTIVE READ CACHE (PRC).....	3
D. PREVIOUS RESULTS	5
E. PURPOSE OF THE STUDY.....	5
F. THESIS ORGANIZATION.....	6
II. ADDRESS TRACES AND SIMULATOR.....	7
A. INTRODUCTION TO TRACE DRIVEN SIMULATION	7
B. TRACE INPUT DATA	7
1. Address Tracing Methods	7
2. BACH (BYU Address Collection Hardware)	8
3. SPARC Trace Data.....	9
C. SIMULATION PROGRAMS	11
1. SACS2	11
2. SACS21	11
3. Modifications for SPARC.....	12
III. SIMULATION RESULTS AND ANALYSIS	13
A. SIMULATION ASSUMPTIONS	13
1. SACS2 Model	13
B. CACHE MODEL PARAMETERS.....	14

1. First-Level Cache Parameters	15
2. Cache Miss Actions.....	15
3. Buffer Parameters.....	17
4. Access Times.....	17
C. SIMULATION RESULTS	18
1. Baseline Testing with No PRC	19
2. Effects of PRC Size on Performance	20
3. Variation in PRC Set Associativity.....	25
4. Variation in PRC Miss Allocation Policy.....	28
5. Variation in the Max Size in Buffer to Continue Read	29
6. Effects of Other PRC Parameters.....	30
D. SECOND-LEVEL CACHE MODELING.....	31
IV. CONCLUSIONS.....	35
A. SUMMARY OF RESULTS.....	35
B. RECOMMENDATIONS.....	36
LIST OF REFERENCES	39
INITIAL DISTRIBUTION LIST	41

LIST OF FIGURES

1. PRC Location in Memory Hierarchy After Ref. [5].....	3
2. Prediction Algorithm From Ref. [5]	4
3. Average Access Times for Various PRC Sizes	24
4. Speedup of System Performance Over a System Without PRC	24
5. Access Times for Various Max PRC Size to Continue.....	30
6. Average Speedup of Three Address Traces Based on Varying Sizes.....	33

LIST OF TABLES

1. Address Traces	10
2. Constant Simulation Model Parameters	16
3. PRC Design Parameters	18
4. Cache Performance Without a PRC.....	20
5. Performance Using 256-Byte 4-way Set Associative PRC.....	21
6. Performance Using 512-Byte 4-way Set Associative PRC.....	21
7. Performance Using 1024-Byte 4-way Set Associative PRC.....	21
8. Performance Using 2048-Byte 4-way Set Associative PRC.....	22
9. Performance Using 4096-Byte 4-way Set Associative PRC.....	22
10. Performance Using 8192-Byte 4-way Set Associative PRC.....	22
11. Performance Using 16384-Byte 4-way Set Associative PRC.....	22
12. Performance Using 32-Kbyte 4-way Set Associative PRC	22
13. Performance Using 64-Kbyte 4-way Set Associative PRC	23
14. Performance Using 128-Kbyte 4-way Set Associative PRC	23
15. Performance Using 256-Kbyte 4-way Set Associative PRC	23
16. Performance Using 512-Kbyte 4-way Set Associative PRC	23
17. Summary of Read Average Access Time and Speedup Using 4-Way Set Associative PRC	23
18. Effects of Changing PRC Set Associativity for 256-Byte PRC.....	26
19. Effects of Changing PRC Set Associativity for 512-Byte PRC.....	26
20. Effects of Changing PRC Set Associativity for 1024-Byte PRC.....	26
21. Effects of Changing PRC Set Associativity for 2048-Byte PRC.....	26
22. Effects of Changing PRC Set Associativity for 4096 Byte PRC.....	26
23. Effects of Changing PRC Set Associativity for 8192-Byte PRC.....	27
24. Effects of Changing PRC Set Associativity for 16384-Byte PRC.....	27
25. Effects of PRC Miss Allocation Policy with 256-Byte 4-way Set Assoc. PRC.....	28
26. Effects of PRC Miss Allocation Policy with 4096-Byte 4-way Set Assoc. PRC.....	28

27. Effects of PRC Miss Allocation Policy with 8192-Byte 4-way Set Assoc. PRC.....	28
28. Effects of Various Max Size in Buffer to Continue Read.....	29
29. Effects of Various UsePRCONWriteMiss	31
30. Effects of Various DropPRCONSecondMiss.....	31
31. Performance for Various Second-Level Cache Sizes.....	32

I. INTRODUCTION

A. MEMORY HIERARCHY

Memory bandwidth and latency are two of the main problems effecting the performance of modern computer architectures. Memory speed should grow linearly with CPU speed to maintain a balanced system. However, in recent years, CPU speed has increased very rapidly and DRAM (Dynamic Random Access Memory) technology cannot meet the needs of fast processors because of latency and bandwidth problems. SRAM (Static Random Access Memory) memories are fast but very expensive and they cannot yet satisfy the cost/performance trade-off. One common approach is to design a memory subsystem which uses both memory types in a hierarchical architecture. This can provide a remarkable speed-up.

A memory hierarchy may consist of many levels but management always occurs between two adjacent levels. The upper level, the one closer to the processor, is usually smaller and faster than the lower level. Performance is the major reason for having a memory hierarchy, thus the speed at which hits and misses are processed is very important [Ref. 1].

The success or failure of an access to the upper level is designated as a hit or a miss. A *hit* is a memory access found in the upper level, while a *miss* means it is not found in that level. Similarly, *hit rate* is the fraction of memory accesses found in the upper level while *miss rate* is the fraction of memory accesses not found in the upper level. The speed can be expressed by hit time and miss penalty. *Hit time* is the time to access the upper level of memory hierarchy, which includes the time to determine whether the access is a hit or a miss. *Miss penalty* is the time to replace a block (the minimum unit of information in the memory hierarchy) in the upper level with the corresponding block from the lower level, plus the time to deliver this block to the requesting device (normally the CPU). The miss penalty is further divided into two components. *Access time* is the time to access the first word of a block on a miss and *transfer time* is the additional time required to transfer the

remaining words in the block. Access time is related to the *latency* of the lower-level memory, while transfer time is related to the *bandwidth* between the lower-level and upper-level memories [Ref. 1].

Combining the miss penalty with the hit rate yields the average memory access time, $t_{\text{access}} = \text{HitRate} \times t_{\text{hit}} + \text{MissRate} \times t_{\text{miss}}$, where t_{access} = the average memory access time, t_{hit} = hit time, and t_{miss} = miss penalty. Therefore, average memory access time serves to summarize the performance of the entire memory hierarchy and more accurately reflects the impact of cache and memory hierarchy design decisions on overall system performance [Ref. 2].

B. CACHE MEMORIES

In the memory hierarchy just described, *cache memories* can be placed in the upper level. Caches are smaller and faster than main memory and are designed to have a high hit rate. Even though caches can only store a small fraction of the data in the main memory, they can contain a significant portion of the microprocessor memory accesses by utilizing the *locality of references* principle exhibited by most programs.

In the 1960s, researchers at IBM discovered that nearly all programs were extremely iterative in nature, a fact which can be taken advantage of in memory subsystem design. If often-needed instructions and data can be stored in a small, very high speed memory, then wait states can be limited only to the less repetitive portions of the program which can be stored in slower, less expensive memory. Using the principles of spatial and temporal locality, the portion of a program that is repetitive can be determined. [Ref. 3].

Most microprocessors today contain a small internal cache which is made of SRAM. Use of a cache memory has become so critical to the operation of modern microprocessors that it has become mandatory to incorporate one directly on the same chip as the microprocessor [Ref. 4].

Microprocessors usually contain on-chip caches within the size range of 2K to 64K bytes. External caches are used both in systems which have and do not have on-chip caches. External caches usually range from 32K bytes to 512K bytes. These caches are made using

slower, more economical SRAM chips. If the processor has an on-chip cache, any external caches used will usually be implemented with architectures which differ from the architecture of the on-chip cache, both to reduce costs and to make up for some of the deficiencies of the on-chip cache [Ref. 3].

C. PREDICTIVE READ CACHE (PRC)

The Predictive Read Cache has been proposed by Fouts and Billingsly [Ref. 5] to reduce the miss penalty disadvantage of cache memories. It is a simple and inexpensive type of cache intended as an alternative to expensive second-level caches. The PRC can have a better performance than larger, second-level caches by using the predictive algorithm described by Fouts [Ref. 5].

Most RISC processors have separate on-chip instruction and data caches. The instruction caches usually have sufficient performance. The PRC focuses on improving the effectiveness of the data cache. Its logical position in the memory hierarchy is in between the data cache and the main memory, as can be seen in the Figure 1.

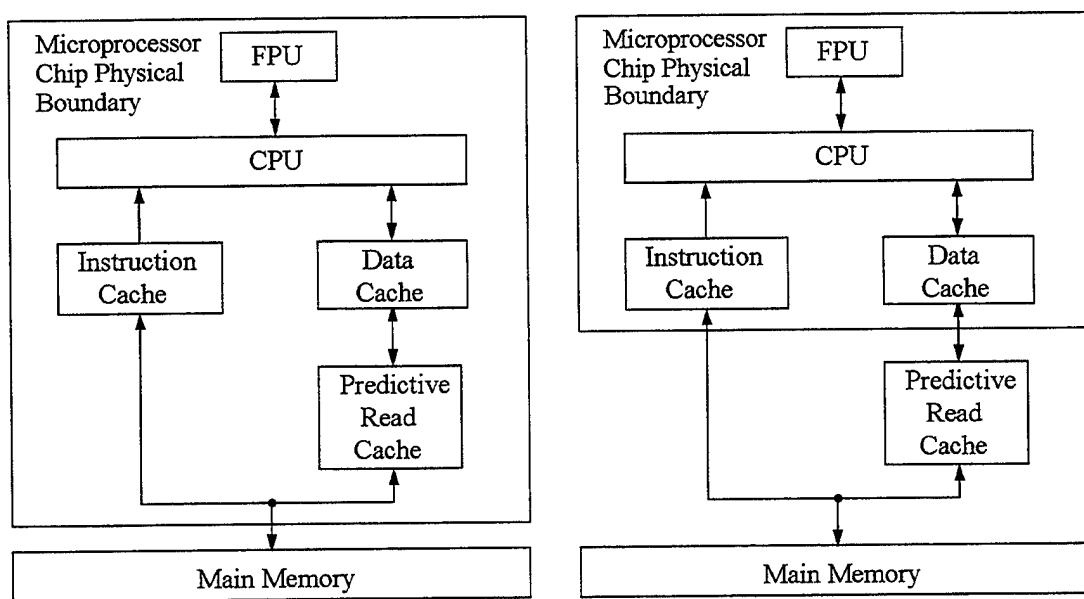


Figure 1. PRC Location in Memory Hierarchy After Ref. [5]

As mentioned above, a PRC is for use with data caches and its function is to predict the next data cache read miss address, prefetch the read data from main memory, and have the data readily available, ready to load into the data cache, when the next read miss in the data cache occurs. When the designers of the PRC explain the PRC operation along with its displacement-based prediction algorithm in [Ref. 5], they address the locality of references principle: in a typical multitasking environment, groups of data references exhibit a strong spatial locality with temporal interleaving.

The prediction algorithm stems from the idea of “the displacement between the next (predicted) address and the current reference address should be the same as the displacement between the current address and the last reference address fetched” [Ref. 5]. The algorithm requires special hardware which can store the most recent miss address (MRMA) and the previous miss address (PRMA) into registers, and then add the difference between MRMA and PRMA onto MRMA to determine the next data read miss as can be seen in Figure 2. This requires the PRC to maintain multiple prediction traces so that task switches will not invalidate the prediction efforts.

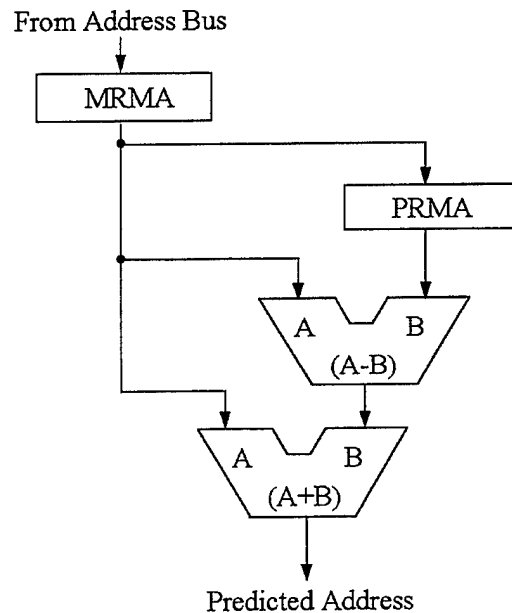


Figure 2. Prediction Algorithm From Ref. [5]

The existing prediction algorithm has some problems involving context switches. Research for a better algorithm is still in progress.

Although Fouts observed that the performance improvement obtained by using a predictive read cache can vary over a fairly wide range (i.e., speedup between 4.42% and 23.33%), it has a very low implementation cost and it can be easily implemented as a part of the microprocessor chip, unlike the second-level cache. It is also ideal for power or chip area limited systems because it uses much less hardware than a second-level cache.

D. PREVIOUS RESULTS

In order to determine the effectiveness of a PRC and to properly optimize its parameters, a set of cache and memory hierarchy simulations was required.

The SACS2 and SACS21 programs were used as cache and memory hierarchy simulators that provide hit and miss percentages as well as the average memory access time. The effects of the PRC were obtained by comparing access times between configurations with and without a PRC and a second-level cache. This reveals the comparative effectiveness of a PRC to the other options. The optimum PRC parameters are indicated by the minimum average memory access time.

The simulations performed by Miller, using trace data from an Intel 486 processor running the SPEC benchmarks, show that a small predictive read cache can give a performance improvement equivalent to a much larger second-level cache [Ref. 2]. For comparison reasons, the details of results obtained by Miller will be discussed in Chapter III, as well as the results of new simulations using trace data from a SPARC processor running the SPEC benchmarks.

E. PURPOSE OF THE STUDY

Efforts to improve the predictive read cache are still continuing to overcome existing problems in order to achieve better performance.

Previous studies, using trace data from an Intel 486 processor running the SPEC benchmarks, have shown that a small Predictive Read Cache can give an excellent

performance improvement equivalent to a much larger second-level cache [Ref. 2]. However, these simulations using Intel 486 address traces, are not representative of contemporary RISC-based microprocessor behavior because of the Intel 486 CISC architecture. On the other hand, since SPARC is the most common RISC architecture today, it is of great importance to simulate the PRC by using address traces captured from a SPARC-based platform.

As a next step, SPARC address traces produced by Brigham Young University [Ref. 7] were used for the simulation of the predictive read cache on the SACS2 simulation program specifically written for PRC [Ref. 2].

F. THESIS ORGANIZATION

This thesis addresses the PRC performance on SPARC-based platforms. Chapter II begins with a review of trace driven simulations and trace data as well as the simulation programs. Chapter III discusses the simulation results and analysis. A summary and conclusions are contained in Chapter IV.

II. ADDRESS TRACES AND SIMULATOR

A. INTRODUCTION TO TRACE DRIVEN SIMULATION

Studies have been initiated to study the performance of the PRC as well as to develop new prediction algorithms. During these studies, one of the most popular performance evaluation techniques, called trace-driven simulation, is being used because it reduces the simulation time dramatically. *Trace driven simulation* is a technique in which a record of events from an actual program execution, a trace, is used as the input for the simulator [Ref. 8]. Of primary importance is understanding the address tracing methods, trace data formats, and the simulation program used. These will be discussed in this chapter.

B. TRACE INPUT DATA

1. Address Tracing Methods

Memory reference traces are often used to evaluate design decisions in memory hierarchies. Determining the characteristics of existing systems running selected workloads requires data about program execution which can be captured by using different monitoring techniques. Most tracing techniques in common use do not capture 100 percent of all references made in the execution of a traced workload. Some approaches may lack the references made by interrupt service routines or exceptions while others may lack operating system references altogether. Tracing techniques also differ by the extent to which they disrupt the sequence of asynchronous events in the traced system. This disruption may result from temporal dilation, increasing execution time, or spatial dilation, increasing program size [Ref. 9].

Trace generation techniques can be classified into three major categories, including software, microcode and hardware approaches. The trade-off regarding complexity versus accuracy among monitoring techniques should be considered with respect to the context in which the data will be used [Ref. 7].

Software techniques involve instruction modification, single stepping and emulation. Although recording references generated by user code is relatively easy when using software techniques, it is very difficult to capture all references, particularly those made by the operating system. It is also very difficult, and sometimes impossible, to conduct accurate tracing of I/O and multitasking workloads because the overhead of the software trace mechanism changes the relative timing of asynchronous events in the system [Ref. 10].

Modifying the microcode to record information about references is another address trace generation technique. This technique suffers from size limitations of the available control store and time dilation, as mentioned above, and generates shorter trace samples. The major drawback is that the microcode technique is not applicable to microprocessor-based systems because its limited to machines with writeable control stores [Ref. 8].

Of all the monitoring techniques, hardware monitoring is the most accurate way of capturing address traces. Although the complexity of the data acquisition system and storage buffer limitations for holding the trace data are the main disadvantages of this method, its the most effective way of monitoring desired signals without slowing or affecting the normal operation of the system.

2. BACH (BYU Address Collection Hardware)

Grimsud et al. [Ref. 7] have built a hardware monitor called BACH (BYU Address Collection Hardware) for tracing microprocessor-based systems. It is a hardware monitor tool in conjunction with simple support software originally designed to generate address traces for memory hierarchy studies. BACH operates by directly monitoring the CPU chip pins. Therefore, the traces contain CPU-generated memory references. It can be interfaced to a variety of microprocessors, including the Motorola MC68030, Intel i80486DX and SPARC microprocessors.

BACH can be used to trace any workload under any operating environment supported by the target platform. It is capable of generating long and complete traces by

using a technique for halting the traced platform during trace buffer emptying so the resulting trace represents contiguous operation of the traced platform.

The traces collected by BACH contain all references, including user references produced by application software, system call references, references produced by interrupt routines, exception handler references, user and supervisor instruction prefetches, and special bus cycles (interrupts, I/Os etc.).

It is designed to generate very accurate reference traces without the disadvantages of typical tracing mechanisms. Research conducted by BYU has shown that the BACH system suffers from an error rate of less than one bit per 4 million collected references. Diagnostic and verification testing is performed by an integral component of the BACH [Ref. 9].

Overall, BACH address traces are good candidates for PRC trace-driven simulations because they meet required specifications as input to PRC models. In the following section, BACH trace data from a SPARC microprocessor will be discussed.

3. SPARC Trace Data

SPARC address traces have been gathered from a SPARCstation 1+ running the Sun-OS operating system. The BYU trace format includes 96 processor signals per reference, collected by the BACH-SPARC interface.

Following is the information that can be obtained from references in the BYU format [Ref. 7]:

- 32 bits of virtual address from the CPU address bus

- 32 bits of data from the CPU data bus

- 4 bits (IRL0-IRL3) for the external interrupt level

- 4 bits (ASI0-ASI3) for the address space identifier (supervisor/user, instr./data, etc.)

- 2 bits (SIZ0-SIZ1) indicating the size of the transfer

- 1 bit R/D (Read/Write)

- 1 bit HAL (Hold Address Latch)

- 1 bit LDST (Load/Store Cycle Indicator)

- 1 bit MEXC (Memory Exception)

- 1 bit MDS (Memory Data Strobe)

1 bit FHL (Floating-Point Hold)

16 bits internal to BACH for cycle count between references

Since the SPARC chipset in the SPARCstation 1+ relies on an external Memory Management Unit (MMU) to translate virtual addresses (32 bits) from the CPU into physical addresses (36 bits), the captured addresses are virtual. For this reason, the traces reflect the actual program locality rather than the locality after the references are translated into appropriate physical accesses by the MMU.

For the traces which will be used for memory hierarchy studies, the internal (on-chip) cache of the processor has been disabled. This is because it is not desirable for the on-chip cache of the processor to filter out a large majority of the CPU generated references.

Another convenient feature of captured traces is that they do not require any instruction decoding because the SPARC processor does not issue instruction prefetch requests and instruction traces can readily be produced from the generated trace.

#, Name, & Desc.	Machine OS	Length/Size 10 ⁶	Refs Format
SDET with 2 users	SPARC Sun-OS	2127 buff/1.96 GB	byutr
Kenbus with 20 users	SPARC Sun-OS	1396 buff/1.26 GB	byutr
Kenbus with 80 users	SPARC Sun-OS	1549 buff/2.30 GB	byutr

Table 1. Address Traces

SPARC address traces (Table 1) produced by Brigham Young University [Ref. 7] were used for the simulation of the predictive read cache on the SACS2 simulation program specifically written for the PRC [Ref. 2].

SDET and KENBUS benchmarks were chosen for this work because they accurately model real programming environments. These two well-established and accepted multiprogramming benchmark programs are contained in the Standard Performance Evaluation Corporation (SPEC) System Development Multitasking Release 1.1 (SDM 1.1) multiprogramming suite. Address traces in Table 1 are discussed in Section 3-C in detail [Ref. 11].

C. SIMULATION PROGRAMS

To estimate the performance of a system using trace driven simulations, two items are needed: a simulation model and input trace data representing the memory references produced by CPU [Ref. 11]. Trace data has been discussed previously and will be detailed in the following chapter. In this section, the simulation model used for the simulations will be introduced. More detailed information involving model parameters and some specifications of the simulation models will also be discussed in Chapter III.

1. SACS2

Smith [Ref. 12] developed a cache and memory hierarchy simulation program written in C, called SACS (Still Another Cache Simulator). SACS incorporates features that allow cache memory optimization based on the average memory access times. The average memory access time is the most important information that can be obtained from SACS, as well as hit and miss percentages. SACS simulates single-level cache and main memory hierarchy using BYU traces in ASCII formats [Ref. 2].

Miller modified and enhanced the original SACS simulator by building a PRC model on its foundation to run functional tests of the PRC using i486 address traces. This new program, called SACS2, gives the user the ability to vary numerous design parameters to provide for testing of the various cache and PRC design options. Running of simulations with different combinations of design parameter values allows determination of the optimal first-level cache and PRC configuration by comparing the average memory access times and choosing the combination that yields the best results. The traces that have been used during these simulations were in a binary format which also required some other modifications to the code [Ref. 2].

2. SACS21

To allow comparisons of system performance with a PRC to that of a system without a PRC or with a second-level cache, another simulation program was required. A new, modified program (SACS21), was written by Miller to simulate the memory hierarchy of a first-level cache and a second-level cache. SACS21 was written after SACS2 by

making necessary changes in the SACS2 code to model a non-predictive, second-level cache.

3. Modifications for SPARC

There are certain design decisions that were made and built into the simulation programs in designing the cache and PRC modeling. These assumptions do not affect the general behavior of the data cache and PRC or the results obtained. However simulations using SPARC address traces required that some modifications be done to the SACS2 and SACS21 programs to ensure proper operation.

Fouts discussed that a PRC should only be used to predict data references since there were already alternate methods to speed up the retrieval of instruction references [Ref. 5]. Therefore, the modified SACS2 analyzes the address trace data and only operates on the data references contained in it. This required a change in the SACS2 code. Thus the special references used by BACH, that are embedded in the trace data, are disregarded during simulations.

III. SIMULATION RESULTS AND ANALYSIS

A. SIMULATION ASSUMPTIONS

The SACS2 simulator incorporates into its design, some of the necessary assumptions used for the simulations. Other assumptions used are set by changing the values of parameters in the simulator. Because these assumptions have an impact on the results obtained, it is necessary to document them here. Some cache model parameters used in these simulations are kept the same as the parameters used in previous simulations to allow even comparisons. However, some parameters had to be changed because of the use of SPARC address traces. Therefore, a set of simulation runs were done for both SPARC and Intel architectures using the same traced program to confirm proper configuration of the simulation software.

1. SACS2 Model

Certain design decisions about the cache and PRC modeling were made and built into the SACS2 program to perform the simulations detailed in this thesis. These assumptions do not effect the general trend of data cache and PRC behavior and the results obtained should be valid for current cache and microprocessor designs. The built-in assumptions are unchanged from their usage in the original SACS2 and are fully discussed by Miller [Ref. 2]. Therefore, they will not be discussed here in detail. However, some changes were made in the code to ensure the same assumptions were compatible with the new SPARC traces.

Fouts discussed that a PRC should only be used to predict data references since there are already alternate methods to speed up the retrieval of instruction references [Ref. 5]. Therefore, SACS2 analyzes the address trace data and only operates on the data references contained in it. This required a change in the SACS2 code. Therefore, the special references used by BACH embedded in the trace data are disregarded during simulations.

Other assumptions used for previous simulations by Miller are also valid for the simulations used to conduct the testing detailed in this thesis. The PRC is located on-chip

between the first-level cache and the buffers that interface with off-chip main memory. The PRC and first-level cache are connected by enough data lines to pass one complete cache block at a time [Ref. 2].

The model assumes that cache misses go to the read buffer and PRC simultaneously, according to the PRC design parameters of Fouts [Ref. 5]. If the data is then found in the PRC, the associated memory read is canceled. This method is used so that cache misses that are also PRC misses do not take longer than they would without a PRC. This method is achievable in current microprocessor designs and ensures that the presence of a PRC does not slow the system down [Ref. 2].

The scoreboarding technique embedded in the design is consistent with current microprocessor design and gives the best use of limited memory bandwidth by checking the buffer for the presence of PRC read requests. The model assumes that a cache miss read request that goes to the read buffer can check the buffer for the presence of PRC read requests for the same address and cancel those requests. This is done because a PRC read for information that is already in the first-level cache would be redundant and a waste of memory bandwidth [Ref. 2].

There are two assumptions made in designing SACS21 when SACS21 is used to simulate a second-level cache. All first-level cache misses go to the second-level cache and any memory read request must wait until after it has been determined that a miss has occurred in both caches. Additionally, since both caches are assumed to be on-chip, all data arriving from memory read requests is made available to both caches on arrival. This allows the first-level cache to satisfy its memory request as soon as the needed data is read from memory and allows both caches to be updated with the new data block simultaneously [Ref. 2]. These assumptions are embedded in the simulator code and are kept the same as they were in previous simulations done by Miller to set up a comparison baseline.

B. CACHE MODEL PARAMETERS

There are a large number of user-definable parameters in the SACS2 simulator. Some parameters are constants because of the architecture-dependent nature of the

simulations. Before starting the simulations, assumptions had to be made to decide what the constant parameter values should be. The values of the parameters that remain constant for all simulations are summarized in Table 2.

1. First-Level Cache Parameters

The traces used for this work were acquired on a SPARCstation 1+ workstation running the SPARC Sun-OS Release 4.1.2 operating system. Therefore, the first-level cache parameters were set to match those on the SPARCstation 1+ for the first three sets of simulations. Another three sets of simulations were run using i486 first-level cache parameters to obtain a comparison base with the previous simulations of Miller, which used the address traces taken from an Intel 486 microprocessor running a UNIX operating system. A direct-mapped, 65536-byte cache with 16-byte blocks and 4-byte words was chosen as the most accurate model for the SPARCstation 1+. A LRU cache block replacement algorithm and a write-through write policy were used. Additionally, a write-around miss strategy was used since subsequent writes to that block will still have to go to the memory [Ref. 6]. For those simulations with an i486 first-level cache, the first-level cache parameters used in previous simulations by Miller are kept the same.

2. Cache Miss Actions

Every first-level cache miss generates a read request that will completely fill the cache block when "Read Forward" is enabled. The value of "CPU Wait for Writes" in Table 3 ensures the use of a write buffer where any writes to memory can be handled in parallel with more cache accesses. "Search Block Buffer" models for a read request generated by the first-level cache to be satisfied if the data needed is currently in the block buffer because of a previous read request. This can significantly improve performance by reducing the time it takes to retrieve the cache miss data. "Update Read Buffer" provides the ability of a memory write request to check the read buffer for pending read requests and remove any bytes stored in the cache by the write. The size of the read request can be reduced and overwriting of the new data in the cache with older data being read from main memory can thus be prevented.

First-Level Cache Parameters	i486	SPARC
Size	8192 bytes	65536 bytes
Block Size	16 bytes	16 bytes
Sub Block Size	4 bytes	4 bytes
Word Size	4 bytes	4 bytes
Associativity	4 way set associative	direct-mapped
Block Replacement Policy	LRU	LRU
Write Policy	Write Back	Write Through
Write Miss Policy	Write Allocate	Write Around

Cache Miss Actions	
Read Forward	Yes
CPU Wait for Writes	No
Search Block Buffer	Yes
Update Read Buffer	Yes
Remove Read Duplicates	Yes
Remove Write Duplicates	Yes

Buffer Parameters	
Read Buffer Size	8
Write Buffer Size	4
Access in Progress Priority	0
Cache Read Miss Priority	1
Write Priority	2
Read For Write Allocate Priority	3
Write Dirty Cache Sub Blocks Priority	4
PRC Predictive Read Priority	5
Slipped PRC Read Priority	6

Access Times	
Read Cache Access Time	1
Write Cache Access Time	1
Cache Hit/Miss Access Times	0
Cache PRC Access Time	1
Memory Access Time	5
Memory Transfer Time	1
Buffer Cache/PRC Access Time	1

Table 2. Constant Simulation Model Parameters

“Remove Read/Write Duplicates” models the ability for only one request for each cache block to be active at a time. If a new request is generated for data that is already in the buffers, it is merged with the pending requests [Ref. 2].

3. Buffer Parameters

Miller discussed that results of preliminary testing of SACS2 shows that with the addition of a PRC there are many read requests generated. Requests generated by cache misses could be blocked by a full buffer if a small read buffer were used. With the larger buffer, these requests can enter the buffer and be included in the ordering of memory accesses by priority [Ref. 2]. For these reasons, the read buffer size was set at eight requests. Since the PRC does not significantly effect the number of writes to memory, the write buffer size was set at four requests. The design parameters of a cache and PRC memory hierarchy define the buffer priorities. Therefore, the predictive read requests were given the lowest priorities so they would have minimal impact on other cache operations. A current cache miss was given the highest priority because it needs to be satisfied as soon as possible.

4. Access Times

Miller [Ref. 2] discussed that access times actually represent the various timing parameters used in the simulations. The resulting average access times are independent of the actual clock speed since these parameters are given as multiples of the system clock cycle. Access times were chosen to accurately reflect current microprocessor and memory designs. Cache access time is the time it takes for the first-level cache to search its tags and determine if the request is present in the cache. For both reads and writes, a time of one clock cycle for the cache access time was chosen. As is mentioned in Section III.A.1, cache misses go to the read buffer and PRC simultaneously. If the request is a miss, the cache miss time is zero cycles and the associated request is sent to the PRC and to main memory during the current clock cycle. If the request to the PRC results in a PRC hit, the cache PRC access time of one cycle will make the data available to the first-level cache on the next clock period. The buffer cache/PRC access times of one period represent the delay

associated with the transfer of data from a full block buffer to the cache or PRC. The memory access time of five cycles models the current large discrepancy between microprocessor clock rates and memory access times. Finally, the memory access time determines how long after a request is made to memory the first word of the data is read into the block buffer. Once the first word of the request has been read from memory, the following words are transferred, one every clock period.

C. SIMULATION RESULTS

Simulations were conducted to test the performance of the memory hierarchy by varying the PRC design parameters summarized in Table 3. The varying PRC design parameters will be discussed in this section, as well as the results of the simulations for given sets of parameters.

PRC Size
PRC Associativity
Use PRC on Write Miss
Max Size in Buffer to Continue Read
Drop PRC on Second Slip
PRC Block Replacement Policy

Table 3. PRC Design Parameters

Numerous simulations were run using trace data from the BACH trace generation system developed at BYU [Ref. 9]. Three address traces were run for functional testing. The first trace is called SDET with two users. SDET is a system-level benchmark designed to measure the performance of any computer system. SDET represents UNIX/C usage in a research and development environment with multiple concurrent users. [Ref. 11]. The first trace of SDET with two users has a total of 44 million memory references and is referred to in this thesis as SDET2. Data references in SDET2 are 17% of the total references. The second and third traces are called KENBUS benchmarks. The KENBUS benchmark, as distributed by SPEC, is derived from the Monash University Suite for Benchmarking UNIX systems (MUSBUS) version 5.2. MUSBUS benchmarks were designed to represent the simulated user workload provided as input parameters. They can simulate single user or multiple users with various workloads. These workloads are reasonably accurate

characterizations of user activity in a research-oriented software development environment. KENBUS uses UNIX shell scripts to exercise the entire computer system and access data from many varied memory locations while ensuring that realistic amounts of terminal I/O were generated and sent to real serial interfaces [Ref. 11]. The second Kenbus trace with 20 users has a total of 37.2 million references and is referred to as KEN20 in this thesis. Data references in KEN20 are 17% of the total references. The third Kenbus trace with 80 users has a total of 37.4 million references and is referred to as KEN80 in this thesis. Data references in KEN80 are 18% of the total references. These traces were chosen as representative of 20 users (KEN20) and 80 users (KEN80).

1. Baseline Testing with No PRC

To obtain the performance of a first-level cache alone, simulations were run with the PRC disabled. This provided a comparison baseline for further PRC simulations. The results in Table 4 show that, as the user count increases from twenty to eighty for Kenbus traces, the cache hit rates decrease because more user processes result in less performance in the L1-cache. The results show that the cache miss penalty increases the average read access time by an average of 42% over that of a perfect cache where all accesses take one cycle. This value was around 40% for the simulations using trace data from an Intel 486 processor. The 2% difference between these two values is reasonable when one considers that the address traces for simulations run during this research were taken from a SPARCstation microprocessor running a Sun-OS operating system. The address traces used in the previous simulations were taken from an Intel 486 microprocessor running a UNIX operating system. Although the write hit rates are not of concern because the PRC is designed to improve average read access time, the dramatic difference between write hit rates of simulations using an i486 with a L-1 cache and simulations using a SPARCstation 1+ with a L-1 cache can be briefly explained as the difference of write policies of these two different caches. For the simulations of the i486 with a L-1 cache, the write policy is a write-back. The simulations for the SPARC with a L-1 cache, use write-through for the first level caches. Over all the simulations, it is observed that the write average access time was a little

over one clock cycle. These results show that the number of consecutive writes to the memory rarely reaches the limit of the write buffer where write requests can be queued without causing a penalty. When the rate at which the memory can complete writes is less than the rate at which the processor is generating writes, no amount of buffering can help because writes are being generated faster than the memory system can accept them [Ref. 13].

L-1 Cache	Trace	Cache Read Hit Rate%	Read Average Access Time	Cache Write Hit Rate%	Write Average Access Time
Sparc	SDET2	92.0	1.331551	68.0	1.000000
Sparc	KEN20	90.2	1.404338	64.3	1.000000
Sparc	KEN80	87.5	1.512575	63.9	1.000000
Intel	KEN20	87.1	1.565927	90.4	1.000309
Intel	KEN80	82.3	1.750647	89.0	1.000305

Table 4. Cache Performance Without a PRC

2. Effects of PRC Size on Performance

The next set of simulations run involved the addition of a PRC to the memory hierarchy and observation of its effects on performance. Table 5 shows the performance with a 256-byte, 4-way, set associative, PRC added to the hierarchy. As can be seen, the PRC hit rate was approximately 26.5%, indicating the PRC correctly predicted the data request coming from the cache about one out of four times. This value was 20% for the simulations using i486 trace data. Results show that the average hit rate of the PRC in a memory hierarchy for SPARC processors is about 33% better than the hit rate of the PRC in a memory hierarchy for Intel 486 processors. The results of simulations using an i486 first-level cache, shown in Table 5, indicate that the average PRC hit rate was about 21%, which is closer to the value obtained in previous simulations. The addition of this very small PRC results in a performance speedup of between 5.8% and 7.1% for read accesses. This is a significant improvement for the small amount of additional hardware required to implement the PRC. The value of speedup for the previous simulations was between 4.6% and 4.9%. This remarkable difference shows that better performance gain is obtained for the simulations using SPARC trace data. There was negligible change in the performance of

write accesses (i.e., it cannot be observed in the floating point precision limits chosen for tables) since the PRC predicts memory reads and is designed to not effect writes.

L-1	Trace	Cache Read Hit Rate%	PRC Read Hit Rate%	Read Average Access Time	Cache Write Hit Rate%	Write Average Access Time
Sparc	SDET2	92.0	26.9	1.254058	68.0	1.000000
Sparc	KEN20	90.5	27.5	1.311637	64.3	1.000000
Sparc	KEN80	87.7	25.1	1.405042	63.9	1.000000
Intel	KEN20	87.2	22.2	1.459670	90.4	1.000309
Intel	KEN80	82.6	19.7	1.627705	89.0	1.000305

Table 5. Performance Using 256-Byte 4-way Set Associative PRC

The results of the simulations of larger PRC sizes are given in Table 6 through Table 16. Simulations of 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144 and 524288 byte PRCs were run to determine the effect of PRC size on performance. First-level cache parameters were set to SPARCstation 1+ specifications and the other PRC parameters were kept the same. A summary of the simulation results for the various PRC sizes is given in Table 17. Figure 3 shows graphically the access times for varying PRC sizes. The speedup of the system performance over a system without a PRC is shown in Figure 4.

Trace	Cache Read Hit Rate%	PRC Read Hit Rate%	Read Average Access Time	Cache Write Hit Rate%	Write Average Access Time
SDET2	92.0	27.4	1.252588	68.0	1.000000
KEN20	90.5	28.0	1.309614	64.3	1.000000
KEN80	87.7	25.3	1.404056	63.9	1.000000

Table 6. Performance Using 512-Byte 4-way Set Associative PRC

Trace	Cache Read Hit Rate%	PRC Read Hit Rate%	Read Average Access Time	Cache Write Hit Rate%	Write Average Access Time
SDET2	92.0	27.9	1.251136	68.0	1.000000
KEN20	90.5	28.5	1.307949	64.3	1.000000
KEN80	87.7	25.5	1.402959	63.9	1.000000

Table 7. Performance Using 1024-Byte 4-way Set Associative PRC

Trace	Cache Read Hit Rate%	PRC Read Hit Rate%	Read Average Access Time	Cache Write Hit Rate%	Write Average Access Time
SDET2	92.1	28.4	1.249633	68.0	1.000000
KEN20	90.5	29.0	1.306076	64.3	1.000000
KEN80	87.7	25.8	1.401555	63.9	1.000000

Table 8. Performance Using 2048-Byte 4-way Set Associative PRC

Trace	Cache Read Hit Rate%	PRC Read Hit Rate%	Read Average Access Time	Cache Write Hit Rate%	Write Average Access Time
SDET2	92.1	29.0	1.247922	68.0	1.000000
KEN20	90.6	29.6	1.304116	64.3	1.000000
KEN80	87.8	26.1	1.400047	63.9	1.000000

Table 9. Performance Using 4096-Byte 4-way Set Associative PRC

Trace	Cache Read Hit Rate%	PRC Read Hit Rate%	Read Average Access Time	Cache Write Hit Rate%	Write Average Access Time
SDET2	92.1	29.9	1.245312	68.0	1.000000
KEN20	90.6	30.4	1.301502	64.3	1.000000
KEN80	87.8	26.7	1.397299	63.9	1.000000

Table 10. Performance Using 8192-Byte 4-way Set Associative PRC

Trace	Cache Read Hit Rate%	PRC Read Hit Rate%	Read Average Access Time	Cache Write Hit Rate%	Write Average Access Time
SDET2	92.2	30.9	1.242225	68.0	1.000000
KEN20	90.7	31.4	1.298078	64.3	1.000000
KEN80	87.9	27.5	1.393952	63.9	1.000000

Table 11. Performance Using 16384-Byte 4-way Set Associative PRC

Trace	Cache Read Hit Rate%	PRC Read Hit Rate%	Read Average Access Time	Cache Write Hit Rate%	Write Average Access Time
SDET2	92.2	32.7	1.237509	68.0	1.000000
KEN20	90.7	32.6	1.293736	64.3	1.000000
KEN80	88.0	28.6	1.389300	63.9	1.000000

Table 12. Performance Using 32-Kbyte 4-way Set Associative PRC

Trace	Cache Read Hit Rate%	PRC Read Hit Rate%	Read Average Access Time	Cache Write Hit Rate%	Write Average Access Time
SDET2	92.4	35.6	1.229794	68.0	1.000000
KEN20	90.8	34.5	1.287288	64.3	1.000000
KEN80	88.2	30.4	1.381909	63.9	1.000000

Table 13. Performance Using 64-Kbyte 4-way Set Associative PRC

Trace	Cache Read Hit Rate%	PRC Read Hit Rate%	Read Average Access Time	Cache Write Hit Rate%	Write Average Access Time
SDET2	92.6	41.6	1.214051	68.0	1.000000
KEN20	91.0	39.4	1.270938	64.3	1.000000
KEN80	88.4	35.6	1.359162	63.9	1.000000

Table 14. Performance Using 128-Kbyte 4-way Set Associative PRC

Trace	Cache Read Hit Rate%	PRC Read Hit Rate%	Read Average Access Time	Cache Write Hit Rate%	Write Average Access Time
SDET2	92.9	51.5	1.188626	68.0	1.000000
KEN20	91.2	50.1	1.234686	64.3	1.000000
KEN80	88.8	56.8	1.266987	63.9	1.000000

Table 15. Performance Using 256-Kbyte 4-way Set Associative PRC

Trace	Cache Read Hit Rate%	PRC Read Hit Rate%	Read Average Access Time	Cache Write Hit Rate%	Write Average Access Time
SDET2	93.0	59.3	1.169294	68.0	1.000000
KEN20	91.4	61.3	1.198161	64.3	1.000000
KEN80	89.1	64.6	1.238375	63.9	1.000000

Table 16. Performance Using 512-Kbyte 4-way Set Associative PRC

Trace	256	512	1024	2048	4096	8192	16384	32K	64K	128K	256K	512K
SDET2	1.254058	1.252588	1.251136	1.249633	1.247922	1.245312	1.242225	1.237509	1.229794	1.214051	1.188626	1.169294
	5.82%	5.93%	6.04%	6.15%	6.28%	6.48%	6.71%	7.06%	7.64%	8.82%	10.73%	12.19%
KEN20	1.311637	1.309614	1.307949	1.306076	1.304116	1.301502	1.298078	1.293736	1.287288	1.270938	1.234686	1.198161
	6.60%	6.75%	6.86%	7.00%	7.14%	7.32%	7.57%	7.88%	8.33%	9.50%	12.08%	14.68%
KEN80	1.405042	1.404056	1.402959	1.401555	1.400047	1.397299	1.393952	1.3893	1.381909	1.359162	1.266987	1.238375
	7.11%	7.17%	7.25%	7.34%	7.44%	7.62%	7.84%	8.15%	8.64%	10.14%	16.24%	18.13%

**Table 17. Summary of Read Average Access Time and Speedup
Using 4-Way Set Associative PRC**

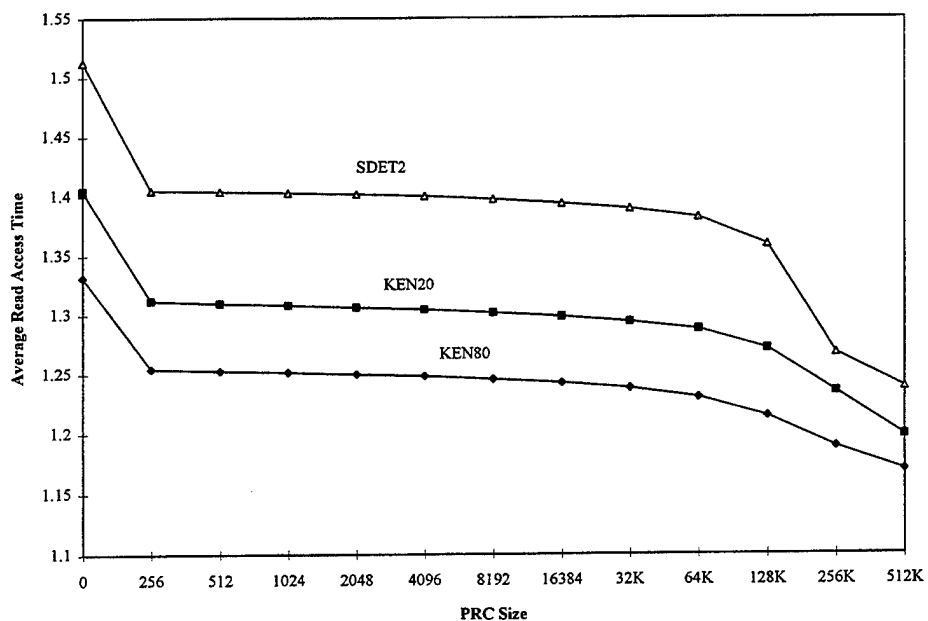


Figure 3. Average Access Times for Various PRC Sizes

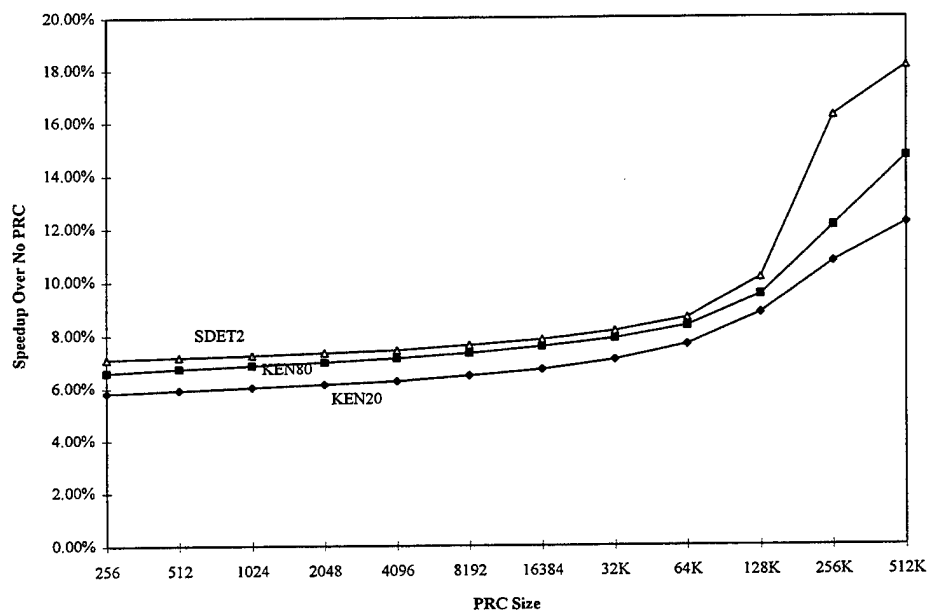


Figure 4. Speedup of System Performance Over a System Without PRC

The results show that a larger PRC results in a higher PRC hit rate and a lower average read access time. Write performance was not significantly effected as PRC size increased, just as observed with the 256-byte PRC size. Miller discussed that the 256-byte, 4-way, set associative PRC can achieve most of the performance gain of the 16384-byte direct mapped PRC [Ref. 2]. The same conclusion can be observed in Figure 3. The higher PRC performance for the larger sizes can be attributed to the PRC retaining its predicted data longer and then acting like a second-level cache to provide the data when needed due to conflict or capacity misses in the first-level cache. However, the small performance improvement as PRC size increases, 6.51% average speedup with a 256-byte PRC increasing to 7.37% average speedup with a 16384-byte PRC, indicates that a PRC 64 times larger only provided an additional 0.86% improvement. For the larger range of PRC configurations, the same comparison shows that as the PRC size increased from 256-bytes to 512 Kbytes, 2050 times larger, the PRC gives only 1.32 times greater performance improvement. These results confirm the previous conclusion that the benefit of the PRC comes mostly from its predictive nature and that the predicted data is normally used by the first-level cache soon after it has been fetched by the PRC. Therefore, the PRC acts to reduce the average memory access time mostly by lowering the penalty for compulsory misses in the first-level cache. The small size of the PRC limits its ability to reduce the penalty for conflict or capacity misses in the first-level cache. Because of this, as long as the PRC has enough locations to hold the different prediction traces, its size has a minimal impact. These results also show that, although the different traces result in different average access times, the trends due to parameter changes are consistent among the traces.

3. Variation in PRC Set Associativity

The next set of simulations performed were to determine the effects of changing the PRC set associativity. Direct mapped and 2 and 8-way associativities were simulated for all PRC sizes, while fully associative PRCs were simulated only for the smaller PRC sizes. Table 18 through Table 24 detail the results of these simulations.

Average Read Access Time					
Trace	Direct Map	2-Way S/A	4-Way S/A	8-Way S/A	Fully Assoc.
SDET2	1.255917	1.254582	1.254058	1.253915	1.254102
KEN20	1.316298	1.313499	1.311637	1.311138	1.311352
KEN80	1.409266	1.406881	1.405042	1.404922	1.405561

Table 18. Effects of Changing PRC Set Associativity for 256-Byte PRC

Average Read Access Time					
Trace	Direct Map	2-Way S/A	4-Way S/A	8-Way S/A	Fully Assoc.
SDET2	1.254165	1.253078	1.252588	1.252379	1.252317
KEN20	1.313720	1.311259	1.309614	1.309402	1.309291
KEN80	1.407834	1.405881	1.404056	1.403943	1.403919

Table 19. Effects of Changing PRC Set Associativity for 512-Byte PRC

Average Read Access Time					
Trace	Direct Map	2-Way S/A	4-Way S/A	8-Way S/A	Fully Assoc.
SDET2	1.252612	1.251679	1.251136	1.251017	1.251885
KEN20	1.311358	1.309139	1.307949	1.307768	1.308295
KEN80	1.406581	1.404632	1.402959	1.402839	1.403618

Table 20. Effects of Changing PRC Set Associativity for 1024-Byte PRC

Average Read Access Time				
Trace	Direct Map	2-Way S/A	4-Way S/A	8-Way S/A
SDET2	1.251063	1.250223	1.249633	1.249575
KEN20	1.308987	1.307129	1.306076	1.305936
KEN80	1.405509	1.403357	1.401555	1.401441

Table 21. Effects of Changing PRC Set Associativity for 2048-Byte PRC

Average Read Access Time				
Trace	Direct Map	2-Way S/A	4-Way S/A	8-Way S/A
SDET2	1.249618	1.248548	1.247922	1.247813
KEN20	1.306947	1.305120	1.304116	1.303927
KEN80	1.404069	1.401885	1.400047	1.399866

Table 22. Effects of Changing PRC Set Associativity for 4096 Byte PRC

Average Read Access Time				
Trace	Direct Map	2-Way S/A	4-Way S/A	8-Way S/A
SDET2	1.247626	1.246295	1.245312	1.244941
KEN20	1.304217	1.302928	1.301502	1.301311
KEN80	1.401052	1.399560	1.397299	1.396946

Table 23. Effects of Changing PRC Set Associativity for 8192-Byte PRC

Average Read Access Time				
Trace	Direct Map	2-Way S/A	4-Way S/A	8-Way S/A
SDET2	1.244808	1.243249	1.242225	1.241714
KEN20	1.300892	1.299107	1.298078	1.297690
KEN80	1.396858	1.395130	1.393952	1.393523

Table 24. Effects of Changing PRC Set Associativity for 16384-Byte PRC

Normally, the advantage of increasing the degree of associativity is that it usually decreases the miss rate [Ref. 13]. Fully associative placement avoids all conflict misses. Associativity is expensive in hardware, however, and may slow access time leading to lower overall performance [Ref. 13]. The same behavior can be observed in the tables above. Increasing the set associativity normally resulted in a small decrease in average access times but the effect varied depending on the size of the PRC, the associativity, and the particular trace being used. A fully associative PRC did not improve the performance and even increased the access time for 256 and 1024-byte PRCs. Overall, there was only a change of 0.08% (i.e. SDET 0.05%, Kenbus (with 20 users) 0.08%, Kenbus (with 40 users) 0.10%) in the average read access time due to associativity changes. The minimal impact of increasing set associativity can be attributed to the fact that conflict misses in the PRC are not common because data read into the PRC is either used quickly by the first-level cache or is not used at all. This leads to the conclusion that the PRC should be designed to minimize hardware complexity and size. For example, direct mapping would normally be preferred.

4. Variation in PRC Miss Allocation Policy

A set of simulations results is shown in Table 25 through Table 27 that were run to determine the effects on average access time of changes in the PRC miss allocation policy using three common cache miss allocation policies (LRU, Random, FIFO). As can be seen, the change among different PRC miss allocation policies resulted in a very small access time variation for 256, 4096, and 8192 byte PRC implementations. The PRC with LRU policy showed the best performance. As is mentioned above, this negligible change of performance for various PRC miss allocation policies is attributed to the fact that data in the PRC is either used soon after it is read in or is not used at all. Therefore, the PRC should use the replacement algorithm that requires the least amount of hardware to reduce system complexity.

Read Average Access Time			
Trace	LRU	Random	FIFO
SDET2	1.254058	1.254868	1.258519
KEN20	1.311637	1.314284	1.324445
KEN80	1.405042	1.406292	1.407632

Table 25. Effects of PRC Miss Allocation Policy with 256-Byte 4-way Set Assoc. PRC

Read Average Access Time			
Trace	LRU	Random	FIFO
SDET2	1.247922	1.249231	1.251682
KEN20	1.304116	1.306400	1.309943
KEN80	1.400047	1.402046	1.405857

Table 26. Effects of PRC Miss Allocation Policy with 4096-Byte 4-way Set Assoc. PRC

Read Average Access Time			
Trace	LRU	Random	FIFO
SDET2	1.245312	1.247156	1.249970
KEN20	1.301502	1.303678	1.307060
KEN80	1.397299	1.399985	1.404000

Table 27. Effects of PRC Miss Allocation Policy with 8192-Byte 4-way Set Assoc. PRC

5. Variation in the Max Size in Buffer to Continue Read

Simulations were run to compare the performance when the *Max Size in Buffer to Continue Read* was varied from 4 to 8 and 16 bytes. The *Max Size in Buffer to Continue Read* parameter determines the maximum portion of the original PRC read that can be remaining and still allow the PRC read to continue. To prevent a cache miss read request from being severely delayed when it enters the read buffer just after a PRC read has started, the simulator has been designed to allow the cache read request to push a PRC read request that is not too far along off the top of the buffer so the cache read request can immediately start. This parameter is critical because, although a PRC read should not hold up a cache read, continually stopping almost complete PRC reads would cause these incomplete reads to fill up the read buffer and no PRC read requests would ever get processed [Ref. 2]. A 256-byte fully associative PRC and a 1024 4-way set associative PRC were used for this set of simulations and results are shown in Table 28 and Figure 5. Changing the size of this parameter to 8 resulted in a slight improvement of the system performance by decreasing the average read access time. Increasing the size to 16, however, resulted in delaying cache read requests and slightly poorer system performance. As can be seen in Figure 5, these results show that the setting of this parameter is a balance between getting cache read misses started as early as possible and not wasting too much memory bandwidth on incomplete PRC read requests. One should examine the system to obtain the optimum results for of this parameter for various implementations of the PRC.

256-byte PRC			
Trace	Max Size = 4	Max Size = 8	Max Size = 16
SDET2	1.254058	1.251371	1.258141
KEN20	1.311637	1.307945	1.312831
KEN80	1.405042	1.398615	1.406986
1024-byte PRC			
Trace	Max Size = 4	Max Size = 8	Max Size = 16
SDET2	1.251136	1.248217	1.254747
KEN20	1.307949	1.304548	1.309027
KEN80	1.402959	1.398615	1.403375

Table 28. Effects of Various Max Size in Buffer to Continue Read

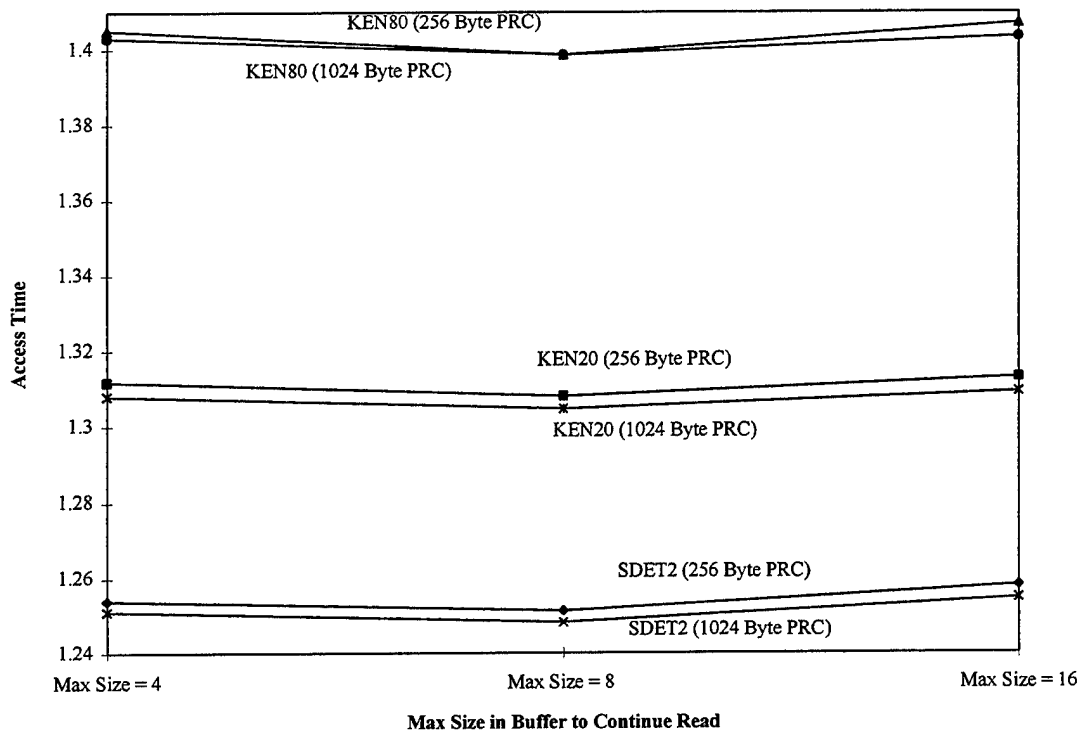


Figure 5. Access Times for Various Max PRC Size to Continue

6. Effects of Other PRC Parameters

UsePRConWriteMiss and *DropPRConSecondMiss* are the last two parameters that will be examined. *UsePRConWriteMiss* = No and *DropPRConSecondMiss* = Yes have been used during all of the simulations so far, based on the results of preliminary testing of previous simulation runs for i486 trace data. Since these parameters could also effect the PRC performance, a set of simulations has been run using a 256-byte fully associative and 1024-byte 4-way set associative PRCs. There was a significant effect on performance for the simulations run previously using i486 trace data when *UsePRConWriteMiss* was set to Yes. Miller discussed this effect as a result of the fact that the writes to memory were not consistent with the data being read. This means that starting a PRC trace based on a write does not provide data that will be used by a future cache read and may even write over a

valid prediction trace [Ref. 2]. As can be seen in Table 29 and Table 30, the performance remained almost the same when the *UsePRConWriteMiss* parameter was varied. This can be attributed to the fact that the write miss policy was set to write around. During a write, the block is modified in the lower level and not loaded into the caches even though *UsePRConWriteMiss* is set to Yes. Consequently, there is no change in system performance.

DropPRConSecondMiss allows the designer to specify whether PRC read requests that have been bumped off the top of the read buffer twice should be canceled or left in the buffer. Changing the *DropPRConSecondMiss* parameter for a 256-byte PRC has a negligible effect on system performance due to the relatively large eight-request read buffer and the mechanism for handling request priorities. A 1024-byte PRC was not affected by this change. Even if a PRC request was slipped twice, it does not impact future reads because it will stay at the bottom of the read buffer until all other read requests are complete [Ref. 2].

Trace	256-byte PRC		1024-byte PRC	
	Use = Yes	Use = No	Use = Yes	Use = No
SDET2	1.254058	1.254102	1.251136	1.251136
KEN20	1.311637	1.311352	1.307949	1.307949
KEN80	1.405042	1.405561	1.402959	1.402959

Table 29. Effects of Various UsePRConWriteMiss

Trace	256-byte PRC		1024-byte PRC	
	Drop = Yes	Drop = No	Drop = Yes	Drop = No
SDET2	1.254058	1.254007	1.251136	1.251136
KEN20	1.311637	1.311190	1.307949	1.307949
KEN80	1.405042	1.405552	1.402959	1.402959

Table 30. Effects of Various DropPRConSecondMiss

D. SECOND-LEVEL CACHE MODELING

Simulations were done using different sizes of second-level caches to allow comparisons of system performance with a PRC to that of systems with a second-level

cache. The results of these simulations performed with the SACS21 simulator are shown in Table 31. As can be observed from the results, increasing the size of a 4-way set associative, two clock-cycle access time, second-level cache gives significant overall system performance improvement. Simulations of 32, 64, 128, 256 and 512-Kbyte second-level caches were performed for comparison with the PRC sizes tested earlier, even though 32-Kbyte and 64-Kbyte second-level caches would not normally be used with a 64-Kbyte first-level cache. Simulations for second-level caches, smaller than or equal in size to the primary cache were run to allow comparisons with very small PRCs.

Size (Kbyte)	SDET2		KEN20		KEN80	
	Average Read Access Time	Speedup	Average Read Access Time	Speedup	Average Read Access Time	Speedup
32	1.283420	3.61%	1.351748	3.74%	1.461859	3.35%
64	1.262393	5.19%	1.333889	5.02%	1.438333	4.91%
128	1.214051	8.82%	1.294941	7.79%	1.365411	9.73%
256	1.140766	14.33%	1.217242	13.32%	1.214376	19.71%
512	1.097383	17.59%	1.157400	17.58%	1.141263	24.55%

Table 31. Performance for Various Second-Level Cache Sizes

Table 31 shows that as the second-level cache size increases the average access times decrease, resulting in better system performance, as is expected. The same result was observed for increasing PRC sizes, as is shown in Table 17. The performance of a 256-byte PRC is always greater than the performance obtained by a 64-Kbyte second-level cache, for all traces. This confirms that the greatest benefit in reducing average memory access time is provided by the predictive nature of the PRC. As the size of the PRC increases, however, the PRC's predictive benefit reduces proportionally to its size. The PRC starts to perform like a second-level cache as its size increases, as is discussed in Section C of this chapter. Finally, for sizes over 128 Kbytes, a second-level cache presents better performance compared to the same size of PRC. This result can be attributed to the fact that unsuccessful predictions cause excessive usage of memory bandwidth, resulting in overall system performance degradation. A 512-Kbyte PRC gives an average speed up of 15.11%,

where the same size of second-level cache results in an average speed up of 19.91%. Figure 6 shows average system speed up for three different benchmarks using varying PRC and second-level cache sizes. Although trade-offs slightly change as the trace input changes, the common behavior of PRCs relative to the second-level caches is essentially unchanged. The second-level cache speedup always increases more rapidly with size than the PRC speedup does.

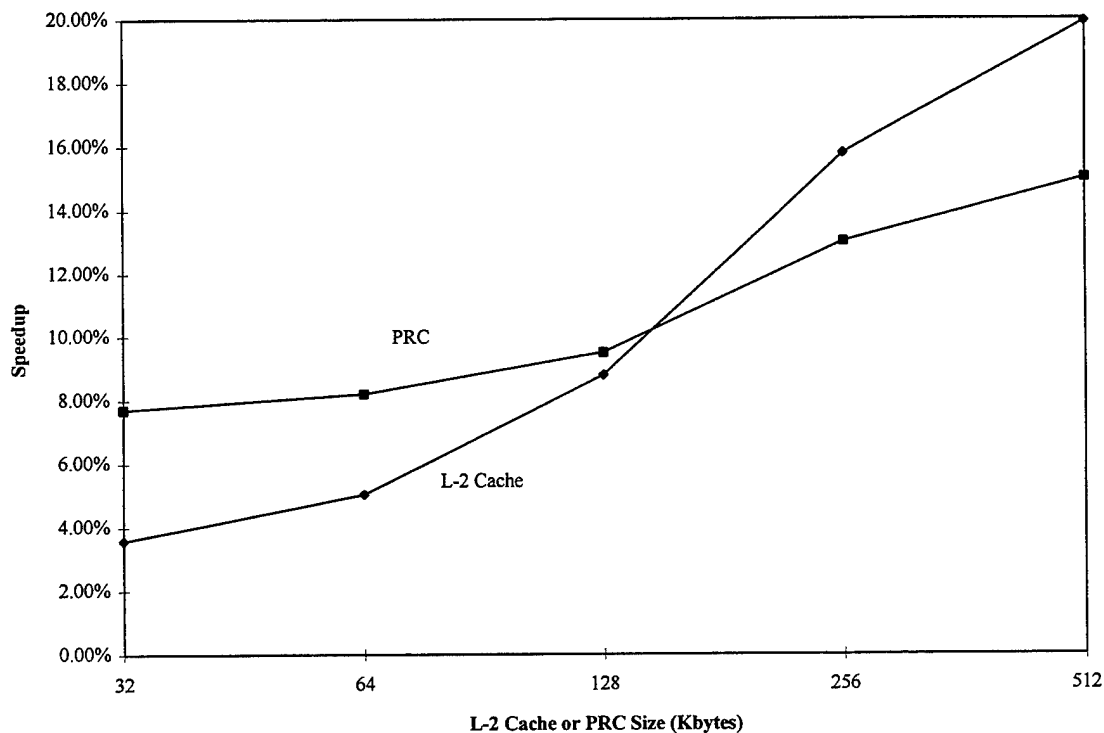


Figure 6. Average Speedup of Three Address Traces Based on Varying Sizes

IV. CONCLUSIONS

A. SUMMARY OF RESULTS

Since SPARC is the most common RISC architecture today, it was of great importance to simulate the PRC by using address traces captured from a SPARC-based platform. The PRC performance simulations presented in this thesis prove that a small PRC can significantly improve the system performance compared to a system with a memory hierarchy containing only a first-level cache. A 256-byte PRC provided a 6.5% speedup while the simulations run previously using i486 trace data and i486 first-level cache provided a 4.7% speedup. When a PRC was added to a first-level cache, the performance improvement of a memory hierarchy of the SPARC was about 38.3% better than the performance improvement of the memory hierarchy of the Intel 486. This is a very promising result when one considers that the SPARC-based platform modeled in this thesis research is representative of contemporary RISC-type microprocessor architecture.

It has also been shown that the performance improvement obtained by a 256-byte PRC was greater than the performance improvement obtained by a 64-Kbyte second-level cache. As the PRC size increases, its predictive advantage decreases proportionally to its size and the PRC starts to perform like a second-level cache. Although the larger PRCs always resulted in a better performance, when very large cache sizes (i.e., over 128 Kbyte) are used, the second-level cache resulted in better performance over the same size PRC. The 512-Kbyte PRC gives an average speed up of 15.11% where the same size second-level cache results in an average speed up of 19.91%. This overall system performance side effect for very large PRC sizes is a result of excessive usage of memory bandwidth, which may stem from unsuccessful predictions performed by the PRC.

As the different PRC configurations were tested, it has been observed that as long as the PRC has enough locations to hold the different prediction traces, its size has a minimal impact. The benefit of the PRC results mostly from its predictive behavior and the predicted data is normally used by the first-level cache soon after it has been fetched by the PRC.

Therefore, the PRC acts to reduce the average memory access time, mostly by lowering the penalty for compulsory misses (the initial filling of the first-level cache) in the first-level cache. Testing the "Max PRC Read in Buffer" parameter showed that proper setting of this parameter is critical. Although a PRC read should not hold up a cache read, continually stopping almost complete PRC reads would waste too much memory bandwidth on incomplete PRC read requests. Therefore, one should examine the system to obtain the optimum setting for this parameter for various implementations of the PRC. For the memory model used in the simulations, varying this value resulted in a slight improvement of system performance. Simulations also showed that varying PRC associativity and PRC block replacement policy (LRU, random, or FIFO) had minimal effect on the performance. Therefore, the PRC should be designed with the easiest method to implement system complexity and hardware cost.

B. RECOMMENDATIONS

The PRC can be used for any kind of processor architecture, including RISC and CISC. It allows the microprocessor systems to operate at high speeds without the need for a second-level, or in some cases primary cache memory. This decreases the amount of required hardware, cost, power consumption, size and weight of high-performance microprocessor systems. An increase in reliability is also expected by using a PRC because a single VLSI chip can replace an entire second level cache. This is of major importance to embedded systems in space-based, weapons-based and portable computing applications.

Different memory hierarchies can be implemented with a PRC and still utilize the benefits of predictive read caches. This is especially true for designs which cannot use a large, second-level cache or even a first-level cache.

A small, predictive read cache can give better performance improvements equivalent to a much larger second-level cache with a very small cost. On the other hand, in the cases where very large predictive read caches are required, it is recommended that the microprocessor system designer use a non-predictive second-level cache.

The PRC can also be implemented in a system as a replacement for the second-level cache or along with a second-level cache. A PRC can also be placed on chip between the first-level and second-level caches or between the first-level cache and main memory.

Efforts to improve the predictive read cache are continuing to achieve better performance. Further studies involve performance evaluation of the predictive read cache and the development of new prediction algorithms.

LIST OF REFERENCES

1. Hennessy, J. L. and Patterson, D. A., *Computer Architecture: A Quantitative Approach*, San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1990.
2. Miller R.W. "Simulation and Analysis of Predictive Read Cache Performance," Master's Thesis, Naval Postgraduate School, Monterey, CA, 1995.
3. Handy, J., *The Cache Memory Book*, San Diego, CA: Academic Press Limited, 1993.
4. Colwell, R. P. and Steck, R. L., "A 0.6um BICMOS Processor with Dynamic Execution," *ISSCC Proceedings*, February 1995, pp. 176-177.
5. Fouts, D. J. and Billingsly, A. B., "Predictive Read Caches: An Alternative to On-Chip Second-Level Cache Memories," *Journal of Microelectronic Systems Integration*, June 1994, pp. 109-121.
6. Catanzaro B., "Multiprocessor System Architectures," 1994 Sun Microsystems, Inc., Mountain View, CA, SunSoft Press.
7. Grimsrud, K. and others, "BACH: A Hardware Monitor for Tracing Microprocessor-Based Systems," unpublished paper, Brigham Young University, Provo, UT, February 1993.
8. Agarwal A., Sites, R. L. and Horowitz, M., "ATUM: A Technique for Capturing Address Traces Using Microcode," In Proc. 13th Int. Symposium on Computer Architecture IEEE, 1986.
9. Grimsrud, K., Archibald, J., Frost, R., Nelson, B. and Flanagan, K., "Estimation of Simulation Error Due To Trace Inaccuracies," Proc. 26th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, 1992.
10. Borg, A., Kessler, R. E. and Wall, D. W., "Generation and Analysis of Very Long Address Traces," In Proc. 17th Int. Symposium on Computer Architecture ACM, 1990.
11. Flanagan J. K., "A New Methodology for Accurate Trace Collection and Its Application to Memory Hierarchy Performance Modeling," Ph.D. Degree paper, Brigham Young University, Provo, UT, December 1993.

12. Smith, W. G., "A Cache Simulator Incorporating Timing Analysis with Buffer and Memory Management, SACS (Still Another Cache Simulator)," Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1994.
13. Patterson, D. A. and Hennessy, J. L., *Computer Organization and Design: The Hardware/ Software Interface*, San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1994.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
2. Dudley Knox Library Naval Postgraduate School 411 Dyer Rd. Monterey, California 93943-5101	2
3. Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5101	1
4. Professor Douglas J. Fouts, Code EC/Fs Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5101	2
5. Professor Frederick W. Terman, Code EC/Tz Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5101	2
6. Deniz Kuvvetleri Komutanligi Personel Daire Bsk.ligi Bakanliklar, Ankara Turkey	1
7. Kara Harp Okulu Komutanligi Kutuphane Bakanliklar, Ankara Turkey	1

8.	Deniz Harp Okulu Komutanligi Kutuphane Tuzla, Istanbul Turkey	1
9.	Hava Harp Okulu Komutanligi Kutuphane Yesilyurt, Istanbul Turkey	1
10.	Golcuk Tersanesi Komutanligi Golcuk, Kocaeli Turkey	1
11.	Taskizak Tersanesi Komutanligi Kasimpasa, Istanbul Turkey	1
12.	Orta Dogu Teknik Universitesi Universite Kutuphanesi Balgat, Ankara Turkey	1
13.	Bogazici Universitesi Universite Kutuphanesi Bebek, Istanbul Turkey	1
14.	Istanbul Teknik Universitesi Universite Kutuphanesi Macka, Istanbul Turkey	1
15.	Dokuz Eylul Universitesi Universite Kutuphanesi Izmir Turkey	1

16. Ltjg Altay Çamlıgüney
1752 Sokak No:2 Sempt Apt. Darie:3
35530 Karşıyaka ,İzmir
Turkey

3